Contents

Ingress Gateway

8 minute read page test

This task shows you how to enforce IP-based access control on an Istio ingress gateway using an authorization policy.

Before you begin

Before you begin this task, do the following:

- Read the Istio authorization concepts.

- Install Istio using the Istio installation guide.
- Deploy a workload, `httpbin` in a namespace, for example `foo`, and expose it through the Istio ingress gateway with this command:

```
$ kubectl create ns foo
$ kubectl apply -f <(istioctl kube-inject -f @s
amples/httpbin/httpbin.yaml@) -n foo
$ kubectl apply -f <(istioctl kube-inject -f @s
amples/httpbin/httpbin-gateway.yaml@) -n foo
```

- Turn on RBAC debugging in Envoy for the ingress gateway:

```
$ kubectl get pods -n istio-system -o name -l i
stio=ingressgateway | sed 's|pod/||' | while re
ad -r pod; do istioctl proxy-config log "$pod"
-n istio-system --level rbac:debug; done
```

- Follow the instructions in Determining the ingress IP and ports to define the `INGRESS_HOST` and `INGRESS_PORT` environment variables.

- Verify that the `httpbin` workload and ingress gateway are working as expected using this command:

  ```
  $ curl "$INGRESS_HOST:$INGRESS_PORT"/headers -s
   -o /dev/null -w "%{http_code}\n"
  200
  ```

> ⚠ If you don’t see the expected output, retry after a few seconds. Caching and propagation overhead can cause a delay.

Getting traffic into Kubernetes and Istio

All methods of getting traffic into Kubernetes involve opening a port on all worker nodes. The main features that accomplish this are the `NodePort` service and the `LoadBalancer` service. Even the Kubernetes `Ingress` resource must be backed by an Ingress controller that will create either a `NodePort` or a `LoadBalancer` service.

- A `NodePort` just opens up a port in the range 30000-32767 on each worker node and uses a label selector to identify which Pods to send the traffic to. You have to manually create some kind of load balancer in front of your worker nodes or use Round-Robin DNS.
- A `LoadBalancer` is just like a `NodePort`, except it also creates an environment specific external load balancer to handle distributing traffic to the worker

nodes. For example, in AWS EKS, the `LoadBalancer` service will create a Classic ELB with your worker nodes as targets. If your Kubernetes environment does not have a `LoadBalancer` implementation, then it will just behave like a `NodePort`. An Istio ingress gateway creates a `LoadBalancer` service.

What if the Pod that is handling traffic from the `NodePort` or `LoadBalancer` isn't running on the worker node that received the traffic? Kubernetes has its own internal proxy called kube-proxy that receives the packets and forwards them to the correct node.

Source IP address of the original client

If a packet goes through an external proxy load balancer and/or kube-proxy, then the original source IP address of the client is lost. Below are some strategies for preserving the original client IP for logging or security purposes.

TCP/UDP Proxy Load Balancer

Network Load Balancer

HTTP/HTTPS Load Balancer

A critical bug has been identified in Envoy that the proxy protocol downstream address is restored incorrectly for non-HTTP connections.

Please DO NOT USE the

⚠ `remoteIpBlocks` field and `remote_ip` attribute with proxy protocol on non-HTTP connections until a newer version of Istio is released with a proper fix.

Note that Istio doesn’t support the proxy protocol and it can be enabled only with the `EnvoyFilter` API and should be used at your own risk.

If you are using a TCP/UDP Proxy external load balancer (AWS Classic ELB), it can use the Proxy Protocol to embed the original client IP address in the packet data. Both the external load balancer and the Istio ingress gateway must support the proxy protocol for it to work. In Istio, you can enable it with an `EnvoyFilter` like below:

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: proxy-protocol
  namespace: istio-system
spec:
  configPatches:
  - applyTo: LISTENER
    patch:
      operation: MERGE
      value:
        listener_filters:
        - name: envoy.listener.proxy_protoco
l
        - name: envoy.listener.tls_inspector
  workloadSelector:
    labels:
      istio: ingressgateway
```

Here is a sample of the `IstioOperator` that shows how to configure the Istio ingress gateway on AWS EKS to support the Proxy Protocol:

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  meshConfig:
    accessLogEncoding: JSON
    accessLogFile: /dev/stdout
  components:
    ingressGateways:
    - enabled: true
      k8s:
        hpaSpec:
          maxReplicas: 10
          minReplicas: 5
        serviceAnnotations:
          service.beta.kubernetes.io/aws-loa
d-balancer-access-log-emit-interval: "5"
          service.beta.kubernetes.io/aws-loa
d-balancer-access-log-enabled: "true"
          service.beta.kubernetes.io/aws-loa
d-balancer-access-log-s3-bucket-name: elb-lo
gs
          service.beta.kubernetes.io/aws-loa
d-balancer-access-log-s3-bucket-prefix: k8sE
LBIngressGW
          service.beta.kubernetes.io/aws-loa
d-balancer-proxy-protocol: "*"
        affinity:
          podAntiAffinity:
            preferredDuringSchedulingIgnored
```

```
DuringExecution:
            - podAffinityTerm:
                labelSelector:
                  matchLabels:
                    istio: ingressgateway
                topologyKey: failure-domain.
beta.kubernetes.io/zone
              weight: 1
      name: istio-ingressgateway
```

For reference, here are the types of load balancers created by Istio with a `LoadBalancer` service on popular managed Kubernetes environments:

Cloud Provider	Load Balancer Name	Lo
AWS EKS	Classic Elastic Load Balancer	TC
GCP GKE	TCP/UDP Network Load Balancer	N

Azure AKS	Azure Load Balancer	N
DO DOKS	Load Balancer	N

You can instruct AWS EKS to create a Network Load Balancer when you install Istio by using a `serviceAnnotation` like below:

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  meshConfig:
    accessLogEncoding: JSON
    accessLogFile: /dev/stdout
  components:
    ingressGateways:
    - enabled: true
      k8s:
        hpaSpec:
          maxReplicas: 10
          minReplicas: 5
        serviceAnnotations:
          service.beta.kubernetes.io/aws-
load-balancer-type: "nlb"
```

IP-based allow list and deny list

When to use `ipBlocks` **vs.** `remoteIpBlocks`**:** If you are using the X-Forwarded-For HTTP header or the Proxy Protocol to determine the original client IP address, then you should use `remoteIpBlocks` in your `AuthorizationPolicy`. If you are using `externalTrafficPolicy: Local`, then you should use `ipBlocks` in your `AuthorizationPolicy`.

Load Balancer Type	Source of Client IP
TCP Proxy	Proxy Protocol
Network	packet source address
HTTP/HTTPS	X-Forwarded-For

- The following command creates the authorization policy, `ingress-policy`, for the Istio ingress gateway. The following policy sets the `action` field to `ALLOW` to allow the IP addresses specified in the `ipBlocks` to access the ingress gateway. IP addresses not in the list will be denied. The `ipBlocks` supports both single IP address and CIDR notation.

ipBlocks remoteIpBlocks

Create the AuthorizationPolicy:

```
$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: ALLOW
  rules:
  - from:
    - source:
        ipBlocks: ["1.2.3.4", "5.6.7.0/24"]
EOF
```

- Verify that a request to the ingress gateway is denied:

```
$ curl "$INGRESS_HOST:$INGRESS_PORT"/headers -s
 -o /dev/null -w "%{http_code}\n"
403
```

- Update the `ingress-policy` to include your client IP address:

ipBlocks remoteIpBlocks

Find your original client IP address if you don’t know it and assign it to a variable:

```
$ CLIENT_IP=$(kubectl get pods -n istio-syst
em -o name -l istio=ingressgateway | sed 's|
pod/||' | while read -r pod; do kubectl logs
 "$pod" -n istio-system | grep remoteIP; don
e | tail -1 | awk -F, '{print $3}' | awk -F:
 '{print $2}' | sed 's/ //') && echo "$CLIEN
T_IP"
192.168.10.15
```

```
$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: ALLOW
  rules:
  - from:
    - source:
        ipBlocks: ["1.2.3.4", "5.6.7.0/24",
"$CLIENT_IP"]
EOF
```

- Verify that a request to the ingress gateway is allowed:

```
$ curl "$INGRESS_HOST:$INGRESS_PORT"/headers -s
 -o /dev/null -w "%{http_code}\n"
200
```

- Update the `ingress-policy` authorization policy to set the `action` key to `DENY` so that the IP addresses specified in the `ipBlocks` are not allowed to access the ingress gateway:

ipBlocks remoteIpBlocks

```
$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: DENY
  rules:
  - from:
    - source:
        ipBlocks: ["$CLIENT_IP"]
EOF
```

- Verify that a request to the ingress

gateway is denied:

```
$ curl "$INGRESS_HOST:$INGRESS_PORT"/headers -s
 -o /dev/null -w "%{http_code}\n"
403
```

- You could use an online proxy service to access the ingress gateway using a different client IP to verify the request is allowed.
- If you are not getting the responses you expect, view the ingress gateway logs which should show RBAC debugging information:

```
$ kubectl get pods -n istio-system -o name -l i
stio=ingressgateway | sed 's|pod/||' | while re
ad -r pod; do kubectl logs "$pod" -n istio-syst
em; done
```

Clean up

- Remove the namespace `foo`:

```
$ kubectl delete namespace foo
```

- Remove the authorization policy:

```
$ kubectl delete authorizationpolicy ingress-po
licy -n istio-system
```